COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-4055

Microsupercomputers: Design and Implementation

Stanford University
Computer Systems Laboratory

Semi-Annual Technical Report

Defense Advanced Research Projects Agency

For the period of April 1989 - October 1989

Contract Number: N00014-87-K-0828

Principal Investigator
John L. Hennessy

Associate Investigator
Mark A. Horowitz

Semi-Annual Technical Progress

April 1989 - October 1989

Contract No. N00014-87-K-0828

Order No. 1133

R & T Project Code: 4331685

This work is supported by the Defense Advanced Research Projects Agency and the Office of Naval Research.

1. Executive Summary

A summary of progress for the period April 1989 through October 1989 follows:

1) *Parallel Architecture:* The Stanford DASH multiprocessor advances the state of parallel computing by combining the programmability of shared-memory machines with the scalability of distributed-memory machines. The key idea on which DASH is built is that of distributed directory-based cache coherence; caches of the processors are kept coherent by sending point-to-point messages between processors on a scalable interconnection network. Our efforts in the recent past have focused on finalizing the design of the DASH prototype. The prototype will consist of 16-64 MIPS R3000/R3010 processors delivering up to 1200 MIPS of processing power. The interconnection network used will consist of a pair of meshes, each with 16-bit wide channels and will use wormhole routing. The design of the cache coherence protocols and the associated state machines is also complete at this point. We expect to have a 16 processor DASH prototype running by Fall 1990.

2) *Parallel Software:* We have developed a compiler algorithm that applies a large set of loop-level optimizations to improve data locality in programs. These optimizations include loop interchange, reversal, skewing, and subblocking. Our technique is unique in that these optimizations are unified into a general transformation, thus we can find the best optimizations without trying every transformation sequences. This new approach to loop transformations can also be used in vectorizing and concurrentizing compilers. We have started on the implementation of our compiler. We have defined and implemented an intermediate format to represent programs that supports experimentation of both scalar and parallelizing optimizations.

3) *Super-Scalar Design:* We have investigated how much parallelism is available at the lowest level -- in the base instruction stream of a processor. Our initial work in this area [1] showed, for the non-scientific applications we are interested in, it is possible to execute a program in about one-half the number of cycles needed by a conventional RISC machine. This work pointed out that the main problem was in the area of instruction fetching and not really contention for functional units.

4) *Multi-level Caches:* The presence of a second-level cache can decrease the optimum size and cycle time of the first-level cache, and significantly improve performance beyond the best attainable with a single level of caching. The optimal characteristics of the second level-cache depend on the miss ratio of the first-level cache, but in general an optimal second level cache will be significantly larger and more likely to be associative than if it were alone in the system. This is because the presence of the first-level cache reduces the number of accesses to the second level without significantly reducing the number of misses in that cache. This shifts the tradeoff away from short cycle times and towards low miss ratios [2].

BiCMOS RAM: During this period we designed a 64K sRAM in TI's .8u BiCMOS technology. The design uses a novel BiCMOS row and column decode that maintain the speed of an ECL diode decode, while reducing the its power dissipation. The RAM is functional with an access time of 4ns, but some design errors have made testing difficult. We are working on correcting these problems.

BiCMOS CAM: Using a new BiCMOS CAM cell we have designed and built a TLB with a 4ns (pad to pad) translation delay. The 64-entry, fully-associative TLB was simply a demonstration vehicle for the CAM cell which uses small ECL-like swings to improve its performance.

Testers: A single chip tester, called Testarossa, contains a dRAM for the test vector storage, a decompressor to increase the effective vector size, and the pin electronics for 16 DUT pins. Each chip can drive 16pins, runs at over 25Mvectors/sec, and can hold about 10KVectors/pin. The chip provides a per-pin architecture, with edge resolution of less than a ns [3, 4]. To build an IMS-class, 256-pin tester using this part would require 16 tester chips and would cost under a few thousand dollars.

Computer Aided Design: In the area of algorithm and tool development for high-level synthesis we have targeted two goals: control generation for synthesized structures and relative scheduling techniques under timing constraints. We have now two high-level synthesis tools: Hercules and Hebe. The former performs behavioral level synthesis into an intermediate form that can be easily simulated. The latter performs user-directed structural synthesis, and it embeds the implementation of the algorithms described before. The Hercules/Hebe tools have been used for two chip designs: a digital audio interface chip (CD or DAT to PC) and a discriminator of a multi-anode photodetector for the space telescope.

We had a major breakthrough in optimal logic synthesis of digital synchronous sequential circuits. We have developed algorithms for minimizing the area of synchronous combinational and/or sequential circuits under cycle time constraints and the cycle time under area constraints. Previous approaches attacked this problem by separating the combinational logic from the registers and by applying circuit transformations to the combinational component only. We have shown instead how to optimize concurrently the circuit equations and the register position. This method is novel and achieves results that are at least as good as those obtained by previous methods. A computer implementation of the algorithms in program MINERVA has been accomplished and experimental results have supported the theory.

Simulation: The goal of this research is to provide application tools for the proposed scalable shared memory multiprocessor. We propose to develop a general purpose simulation environment which serves as an interface between application programs and the underlying machine architecture

and operating system. The operations of the simulation environment will match those of the processors; hence only a minimum amount of overhead will be incurred for synchronization and communications in the course of computation. A generic user interface will be provided for users to specify their application programs, which will cover the domains of scientific computation, behavioral simulations for large complex systems, computer-aided design for fast prototyping, computer networks and communications protocols, and manufacturing and mechanical simulation.

2. Technical Progress:

2.1 Parallel Processor Architecture

Design of Scalable Shared-Memory Multiprocessors

As stated in our previous report, we are currently pursuing the design of a scalable multiprocessor. Our design goals for this machine are: (i) it should be a general purpose machine; (ii) it should be a good target for the work being done on parallelizing compilers and parallel programming languages; and (iii) the architecture should be scalable to support a large number of high-performance processors, so that the resulting machine is significantly more powerful than the latest uniprocessors. Working towards these goals, we are currently building the prototype of a scalable shared-memory multiprocessor, called DASH (Directory Architecture for SHared memory), that supports hardware cache coherence and provides support for dealing with memory latency and synchronization.

At the top level, the DASH architecture consists of a number of processing nodes connected through a high-performance low-latency interconnection network. The physical memory in the machine is distributed among the nodes of the multiprocessor, with both local and remote memory directly accessible to each node. Each processing node, or *cluster*, consists of a small number of high-performance processors with their individual caches, a common cache for the cluster, a portion of the shared memory, and a directory controller interfacing the cluster to the network. A simple bus-based snoopy scheme is used to keep caches coherent within a cluster, inter-node cache consistency is maintained using a *distributed directory-based* cache coherence protocol. In this scheme, each processing node has a directory memory corresponding to its portion of the shared physical memory. For each memory line, the directory memory stores identities of all remote nodes caching that line. Using this information it is possible for a node writing a location to send point-to-point messages to invalidate remote copies of the corresponding cache line. This is in contrast to the invalidating broadcast usually required by the snoopy protocols. The scalability of DASH is greatly enhanced by this ability to avoid broadcasts.

Since our last report, the design of the machine has become quite concrete. We have decided to use the Silicon Graphics 4D/240 workstations as the individual nodes in our multiprocessor. The 4D/240 is itself a multiprocessor consisting of 4 processors, with each processor consisting of a 25MHz MIPS-R3000/R3010 set delivering about 20 MIPS of integer and 4 MFLOPS of floating point performance. The main reason for using a commercial machine as the node in our prototype is to keep the size of the design effort manageable, thus reducing the time to completion of the prototype. It was a complex trade-off, since we do loose some design flexibility, but we felt it was worth it. The interconnection network used in the DASH prototype will consist of a pair of meshes, each with 16-bit wide channels and will use wormhole routing. We have modified the Caltech Mesh-Routing-Chips to suit our needs. The chips have recently come back from fabrication and we are in the process of testing them.

During the past year, we have also been working on finalizing the directory-based cache coherence protocol. We have moved away from the blocking protocol we had originally proposed to a non-blocking protocol. In the non-blocking protocol, when a request for a location that is dirty in a remote cache is received by a directory, the directory forwards that request to the remote cluster rather than holding on to the request. This change reduces the buffering requirements and enhances the performance of the machine. We have also been spending time on the issue of level of consistency for shared-memory in DASH. We have moved away from the traditional notions of sequential consistency and weak consistency to a still weaker form of consistency. We are calling our scheme *release consistency*. The key idea is that the processor needs to wait for its writes to complete only before unlock instructions, and not before and after all synchronization instructions as required by weak consistency. We are currently doing simulation studies to quantify the benefits.

With the design of the protocol finalized, we have been spending considerable time on the design of the directory controller board. The directory controller board will consist of (i) the directory memory, (ii) the intercluster coherence state machines, (iii) the logic needed to support the interconnection network, and (iv) a hardware monitor. The design of the

state machines and the data paths is almost complete at this time. We have also completed the design of the hardware monitor. We are currently awaiting tools from VALID and Xilinx to begin schematic entry.

Finally, we have been spending time looking into operating system issues for the DASH prototype. We will be either extending the current operating system from Silicon Graphics or we will be porting MACH. We are talking to both groups and having discussions with them. We are also exploring implications of some special features that DASH needs, for example, TLB consistency within a multi-cluster environment, double and triple mapping of memory pages to support locks and prefetch mechanisms, and other memory management and process scheduling issues.

Basic Architecture Studies and Simulation Tools Effort

One of our main focuses has been on obtaining multiprocessor memory reference and synchronization traces from "real" parallel applications, and using them for the design of parallel architectures. Our original studies were done using traces obtained from a VAX-8350 (using modified microcode). A major limitation of the microcode-based scheme was that it was not possible to get traces for more than 4 processors. Subsequently, we developed and used a scheme based on the VAX T-bit. This enabled us to use an arbitrary number of processes but was too slow. It took about a week to get a reasonable length 32 process memory reference trace. We now have a new and powerful tool called Tango to help with this task.

Tango is a software tracing and simulation system that provides data to aid in evaluating parallel programs and multiprocessor systems. The system provides a simulated multiprocessor environment by multiplexing application processes onto a single processor. Tango allows the user to trace the shared memory and synchronization behavior of parallel programs. The system is efficient, and can be used with a wide range of machine and programming models. The Tango system currently runs on the MIPS M120 and on the DECStation 3100. It is about 100-1000 times faster than our old T-bit based trace generator, and thus qualitatively changes the kinds of studies that we can do. In general, accurate tracing is difficult since parallel programs are typically non-deterministic; the

execution path through the program depends on the real time behavior of the hardware system. Tango offers accurate tracing by allowing the user to optionally integrate his shared memory and synchronization timing simulations into the tracing environment. We currently have a detailed simulator of the DASH prototype integrated with Tango. This system is being extensively used for studying architectural tradeoffs and also for verifying our directory-based coherence protocol.

We are also continuing our studies about invalidation patterns in multiprocessors. Since our last report, we have gathered additional information about the affect of varying the cache line size on invalidation patterns. Our studies show that the best invalidation behavior is achieved when the cache line matches the size of the data objects being shared. Both line sizes that are too small and line sizes that are too large can drive up the average number of invalidations per shared write. We note, however, that this effect is not very strong. Consequently, it appears that high performance multiprocessors should use a large cache line size (32 - 64 bytes) to benefit from pre-fetching effects and to hide network latency encountered in scalable shared-memory multiprocessors.

2.2 Parallel Software

In the area of compiler research, we have made progress in two directions. First, we have developed an algorithm to improve data locality. Second, we have defined and implemented a compiler intermediate format to support parallelization.

Algorithm to Improve Data Locality

We have developed a new approach to loop-level transformations, and applied it to improving data locality in code. While our current focus is increasing a uniprocessor's cache hit ratio, this technique can be extended also to improving data locality for multiprocessors, a crucial factor in the performance of large-scale parallel systems.

Previous research on loop-level transformations has identified a set of loop transformations that are useful for vectorization and concurrentization;

they include loop interchange, subblocking, quantization, reversal, and skewing. The same optimizations have can also be used to increase data locality in programs. However, in general, little is known about when to apply which of these optimizations. The strategy typically used is one of "generate and test": apply different sequences of stepwise code transformations and evaluate the worth of all unique final designs.

We have discovered that the key transformation that improves locality is subblocking, and that the purpose of all other transformations is only to make it legal to subblock the desirable loops. Therefore the algorithm consists of (1) a procedure to identify the loops to subblock, and (2) a procedure to transform the loops so as to maximize the number of desirable loops subblocked. The former procedure is specific to data locality and is formulated as finding the kernel to a "locality" matrix. The latter procedure, described below, addresses the general problem of finding the right combination of optimizations for a given objective.

Our new approach can find the best program without exhaustively testing all legal combinations of stepwise transformations. We have unified the transformations of loop interchange, reversal and skewing, showing that a single general transformation can achieve the result of all combinations of stepwise optimizations. For example, a series of loop interchange is simply a permutation of the loop ordering. This is significant because we can systematically and effectively search the space of final designs; we avoid exhaustive enumeration by pruning the search space with our evaluation function.

Our current work includes (1) implementing the algorithm and gathering empirical data, (2) extending the data locality algorithm to multiprocessors and (3) applying this new code transformation technique to code parallelization.

Compiler Implementation

The compiler we are constructing is unique in that it integrates both high-level parallelizing transformations and scalar optimizations in a common framework. Existing parallelizing compilers use two separate programs: a

source-to-source compiler performs high-level code restructuring, and a conventional compiler that translates source code into object code. The integration of parallelization and scalar optimizations is important because scalar information can be used for parallelization and vice versa. Many of the standard scalar optimizations such as constant propagation and forward substitution are useful for parallelism detection. Conversely, if we want to exploit instruction level parallelism, this high-level data dependence information must be made available to the machine code scheduler. Moreover, a clean, simple internal representation is more conducive to high level code transformations than the source level representation.

One of the key issues in integrating the parallelizing and scalar optimizations is the intermediate code representation. The former set of optimizations needs higher level information and the latter needs lower level information. We have identified the necessary high-level information to be control constructs and array accessing operations. We have developed a dual code representation that allows different optimizations to view the same information at different levels.

We have gained some experience in using the intermediate format by building several rapid prototypes of code transformations. We have uncovered some weaknesses and have corrected them in our current design. Although we expect the intermediate format to evolve as we develop our optimizations, we believe that we now have the basic functionality to support our experimentation on both scalar and parallelizing optimizations.

2.3 Uniprocessor Architecture

Super Scalars

We have continued our work on investigating how much parallelism is available at the lowest level -- in the base instruction stream of a processor. Our initial work in this area [1] showed, for the non-scientific applications we are interested in, it is possible to execute a program in about one-half the number of cycles needed by a conventional RISC machine. But to achieve

this type of speedup in hardware required branch prediction, a four wide instruction decoder, out-of-order execution, and register renaming [5] -- a non-trivial amount of hardware.

The out-of-order issue allows an instruction *B* that logically follows *A* to be issued before *A* if it does not have any data dependencies. Register renaming increases the available parallelism by removing data dependencies that arise because of storage conflicts, caused by register reuse. To implement these features in hardware is difficult at best, and would require a fair amount of hardware. We are currently investigating methods of moving these steps out of the hardware and into the compiler system. The question we are trying to answer is what percentage of the available parallelism can a software system capture. Our approach is to change the hardware slightly to allow the software to move code through branches, giving the compiler more opportunities to optimize the code. At present we have the basic compiler up and running, and have a system to estimate the performance advantage of moving code through branches. We also have a proposal for the basic machine organization. Our next step is to complete the work on the software, and if the results look promising to flesh out the machine details.

2.4 Computer Aided-Design

High-level Synthesis.

In the area of algorithm development for high-level synthesis we have targeted two goals: control generation for synthesized structures and relative scheduling techniques under timing constraints. The former problem is the one of generating control that achieves the highest possible performance (in terms of number of cycles) for arbitrarily nested structures and in the presence of non-deterministic delays. Arbitrary nesting at no performance cost allows for the unlimited use of procedure calls in hardware description languages for synthesis without having to expand them in the synthesis phase. Allowing for non-deterministic delays corresponds to supporting synchronization and data-dependent iteration primitives. The latter problem allows for the well-posedness and validity check of timing bounds of the number of cycles taken by sequences of

operations in the presence of non-deterministic delays as well as the construction of a valid schedule.

As far as tool development is concerned, we have now two tools: Hercules and Hebe. The former performs behavioral level synthesis into an intermediate form that can be easily simulated. The latter performs user-directed structural synthesis, and it embeds the implementation of the algorithms described before. The Hercules/Hebe tools have been used for two chip designs: a digital audio interface chip (CD or DAT to PC) and a discriminator of a multi-anode photodetector for the space telescope.

Logic Synthesis

In this area we had a major breakthrough in optimal logic synthesis of digital synchronous sequential circuits. We have developed algorithms for minimizing the area of synchronous combinational and/or sequential circuits under cycle time constraints and the cycle time under area constraints. Previous approaches attacked this problem by separating the combinational logic from the registers and by applying circuit transformations to the combinational component only. We have shown instead how to optimize concurrently the circuit equations and the register position. This method is novel and achieves results that are at least as good as those obtained by previous methods. A computer implementation of the algorithms in program MINERVA has been accomplished and experimental results have supported the theory.

2.5 Simulation

Parallel simulation has been proposed to explore the potentials of the DASH multiprocessor machine for CAD applications. Integration of existing simulators (THOR, IRSIM, and SPICE) for mixed-mode circuit and system simulation has been developed to provide a parallel multi-level simulation environment. Parallelism is obtained within each simulator by decomposing its simulation into smaller blocks and among the simulators by providing a communication and synchronization interface between them.

At a first stage, algorithms for distributed and parallel simulation were investigated. Research has been conducted on extensions to the known time-stamped algorithms by Chandy and Misra. These extensions consist of adding additional information to time-stamped messages: besides the time of an event, an interval of duration and a label to uniquely identify events are added in each message, which would help to reduce deadlock occurrence, increase concurrency, and reduce communication traffic overhead.

A prototype that integrates VLSI CAD simulators to test and examine these concepts is under development. The simulation environment serves as a printed circuit board backplane where existing simulator programs are plugged in to run in parallel either at the same design level or at different design levels to function as an harmonic parallel mixed-level simulator. Basically the simulation environment consists of a kernel to handle the interface between simulators and to coordinate their computation. Also the environment handles conversion between design levels as in the case of logic signals and circuit signals. The simulators being integrated consist of THOR, a behavioral simulator for use with system design at either the functional and register transfer level, IRSIM, a switch level timing simulator for digital circuits, and SPICE, a general-purpose circuit simulator.

The prototype system is being implemented on a network of DEC workstations and will be transported to a multiprocessor machine after the programs have been tested. The design of a high rate adaptive filter consisting of more than 30 chips will be the first example that we use to measure the simulation performance. Three types of simulators at the behavioral level, the switch level, and the circuit level will be necessary to be incorporated into the simulation as we need both custom-design and commercial chips to implement the filter system. The speedups obtained from using a multiprocessor for mixed-mode simulation will be evaluated.

2.6 VLSI Design

We have been continuing our efforts to exploit the capabilities of integrated circuit technologies for high speed systems. During this period we have continued our effort BiCMOS, testing and high-speed arithmetic.

BiCMOS

Our work in BiCMOS has continued in three fronts: fast sRAMs, innovative BiCMOS logic circuits, and CAD tools for BiCMOS. In the area of fast sRAM we were very pleased to find out that our new BiCMOS memory cell design, the CSEA cell, was used by a commercial company, Aspen, in their 3ns 4K BiCMOS RAM [6].

In this past period we designed a 64K bit sRAM. The RAM was designed in in Texas Instruments' 0.8μ BiCMOS technology, and was fabricated by TI. The sRAM was designed to demonstrate it is possible to build a large, high-speed (under 4ns) BiCMOS sRAM, while maintaining a reasonable power dissipation (1.5W). It uses the CSEA cell, with a bipolar transistor in each memory cell that we reported earlier. The design uses an innovative address decoder that combines the high-speed of a standard diode decoder, while greatly reducing the power that the decoder requires. The power reduction is accomplished by replacing the resistor load in a diode decoder with a pMOS device, and then using the gate of the pMOS to control its resistance. This technique allowed us to reduce the decoder power by a factor of 4, which reduced the power of the part by about a factor of 3. The sRAM also used a novel write-path. Instead of having a set of CMOS decoders for writes, the RAM uses the read decoders and provides a ECL-CMOS converter per worldline. Sharing the decoder increases the write speed, while reducing the complexity of the part.

The parts have returned from fab and we are currently testing and debugging them. Unfortunately the design needed to be finished in 2 months to meet the fab deadline. As a result there are three design errors that we have found. By probing the design we have confirmed the basic operation of the RAM, and using a focused-ion beam machine from Seiko we have partially repaired on chip. The chip has a read access time of 4.5ns, which is slower than expected. We expect to have better results soon.

In addition to the 64K RAM, during this last period we have also come-up with a new BiCMOS CAM cell. To demonstrate the features of the CAM, we designed a 64 entry, fully associative TLB in TI's BiCMOS technology. The CAM uses small signal swings on all the critical paths. The experimental data shows a 4ns translation delay, from virtual-address in to translated address out. Removing the pad delay, the internal speed of the device is on the order of 3ns. This CAM should be useful for high-speed computers.

The last major part of our BiCMOS effort is in creating CAD tools to support BiCMOS designs. We are using Magic for layout, but simulation is a much more difficult problem. To try fill the current gap in simulation tools for BiCMOS circuits we are working on Bisim, an Rsim like simulator for Bipolar and BiCMOS circuits. Like Rsim Bisim is a switch level simulator, but unlike Rsim, Bisim understands that signals are voltages rather than simple boolean values. This extra flexibility allows Bisim to handle a wider class of circuits than Rsim can handle. In particular it should be able to correctly simulate a wide class of sense circuits -- circuits that often occur in BiCMOS. At this point we model both MOS and bipolar devices by piecewise linear devices, and have written code that will find the final values for networks of these devices. These algorithms have been incorporated in Bisim, and we are now evaluating their performance by trying to simulate the BiCMOS sRAM we designed. The initial results look promising, but we have a significant amount of work still to do. Although we have simple timing models in this version of the simulator, it seem like these models might need some improving.

Integrated Testers

During this period we have continued to work on building a single chip tester. The chip, called Testarossa, contains a dRAM for the test vector storage, a decompressor to increase the effective vector size, and the pin electronics for 16 DUT pins [7]. We have received the 2nd run of these chips. This version was fabricated in a 1.6μ CMOS technology. These chips were fully functional, and preliminary testing indicates that we have enough fully functional parts to build a 256 pin, 25 MHz tester.

During this period we have been working on getting the software for the tester working, and working on designing the other component needed by the tester. The chip has three delay generators per pin (a total of 48 in a chip) so to make the circuits smaller there provide fine resolution, but low accuracy. To set the delays accurately the system needs one precise time reference, and then uses internal comparators to adjust the delay to make the external reference. For the testing so far, we have used a set of HP pulse generators. We are currently designing a CMOS chip to take this function. It will contain a PPL and some precision delay generators (generated using closed-loop feedback) to provide the clocks needed for the tester. We hope to have this chip out to fabrication in 3 to 6 months.

High-Speed Arithmetic

We had previously shown how to build dense, fast multipliers by building a partial multiplier tree, and pipeline the tree after every two carry-save adders. Then by clocking the structure quickly (100s of MHz) one can complete a multiply only slightly slower than a full tree. Recently we have shown how to perform IEEE rounding in this type of structure, and high-speed multipliers in general [8]. It turns out there is a way of starting the final carry propagate add before the carry-in is known. This result is very important since in iterative multipliers, the carry-in is only known 1 to 2 cycles after the rest of the result has been generated. Using this rounding methods allows one to perform IEEE compatible multiplication nearly as fast as producing a truncated result. The overhead in hardware is also not too large, less than 25% added hardware.

We are again using division to test out some of our ideas in self-timed design. Our goal is to build an iterating divider that runs as fast as the combinational logic allows -- it is no slower than building the complete array in hardware. To achieve this result we need to remove the latches from the critical path (they would add delay) and insure that the control signals never slow down the computation. We are calling this self-timed design style "No Overhead Logic". Our initial logic design work looks promising, and we are tweaking the device sizes and timing before we go to layout. We hope to have the design done by the end of the school year.

3. Publications, Presentations, Reports

1. Smith, M., Johnson, M. and Horowitz, M., Limits on Multiple Instruction Issue, ACM/IEEE, *ASPLOS-III*, 290-302, Boston, MA. April, 1989.

2. Przybylski, S., Horowitz, M. and Hennessy, J., Characteristics of Performance-Optimal Multi-Level Cache Hierarchies, IEEE/ACM, *16th International Symposium on Computer Architecture*, Jerusalem, Israel. June, 1989.

3. Gasbarro, J. and Horowitz, M. Integrated Pin Electronics for VSLI Functional Testers. *IEEE Journal of Solid State Circuits*. April, 1989.

4. Gasbarro, J. and Horowitz, M., Integrated Pin Electronics for VLSI Functional Testers, IEEE, *Custom Integrated Circuits Conference*, 16.2.1-16.2.4, Rochester, NY. May, 1988.

5. Johnson, W. *Super-Scalar Processor Design*. Ph.D., Stanford University, June, 1989. Also appears as CSL technical report no. 89-383.

6. Cole, B. Aspen Shows BiCMOS Can Yield Fast SRAMs. *Electronics*. 88, February, 1989.

7. Gasbarro, J. and Horowitz, M., Testarossa: A Single-Chip, Functional Tester for VLSI Circuits, IEEE, *International Solid-State Circuits Conference*, San Francisco, CA. February, 1990.

8. Santoro, M., Bewick, G. and Horowitz, M., Rounding Algorithms for IEEE Multipliers, IEEE, *9th Symposium on Computer Arithmetic*, 176-183, Santa Monica, CA. September, 1989.

9. Acken, J., Agarwal, A., Gulak, G., Horowitz, M., McFarling, S., Richardson, S., Salz, A., Simoni, R., Stark, D. and Tjiang, S. "The MIPS-X RISC Microprocessor." Chow ed. 1989 Kluwer Academic Publishers. Boston, MA.

10. Agarwal, A., Hennessy, J. and Horowitz, M. Cache Performance for Operating Systems and Multiprogramming Workloads. *ACM Transactions on Computer Systems*. 1988.

11. Agarwal, A., Horowitz, M. and Hennessy, J. An Analytic Cache Model. *ACM Transactions on Computer Systems*. 7, (2): 184-215, May, 1989.

12. Aho, A., Ganapathi, M. and Tjiang, S. Code Generation Using Tree Matching and Dynamic Programming. *Transactions on Programming Languages and Systems*. October, 1989.

13. Blatt, M., Yield Evaluation of a Soft-configurable WSI Switch Network, Plenum Press 1990, *1989 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, Tampa, Florida. October, 1989.

14. Chandra, R., Gupta, A. and Hennessy, J. L., COOL: A Language for Parallel Programming, Pitman, *Proceedings of the Second Workshop on Languages and Compilers for Parallel Computing*, University of Illinois at Urbana-Champaign. August, 1989.

15. De Micheli, G. and Klein, T., Algorithms for Synchronous Logic Synthesis, IEEE, *International Symposium for Circuits and Systems*, Portland, OR. May, 1989.

16. Ganapathi, M. Prolog Based Retargetable Code Generation. *Computing Languages*. 14, (3): 193-204, February, 1989.

17. Ganapathi, M. and Kennedy, K., *Interprocedural Analysis and Optimization*, Rice University, Technical, COMP TR89-96, July, 1989.

18. Ganapathi, M. Semantic Predicates in Parser Generators. *Computing Languages*. 14, (1): 25-33, October, 1989.

19. Ganapathi, M. and Mendal, G. Issues in Ada Compiler Technology. *IEEE Computer*. 22, (2): 52-60, February, 1989.

20. Gupta, A., Forgy, C., Kalp, D., Newell, A. and Tambe, M. Parallel Implementation of OPS5 on the Encore Multiprocessor: Results and Analysis. *International Journal of Parallel Programming*. 17, (2): 1988.

21. Gupta, A., Forgy, C. L., Kalp, D., Newell, A. and Tambe, M., Parallel OPS5 on the Encore Multimax, *Proceedings of International Conference on Parallel Processing*, 1988.

22. Gupta, A., Forgy, C. and Newell, A. High-Speed Implementations of Rule-Based Systems. *ACM Transactions on Computer Systems*. 7, (2): 119-146, May, 1989.

23. Hennessy, J. L. and Patterson, D. A. "Computer Architecture: A Quantitative Approach." 1990 Morgan Kaufmann Publishers, Inc. San Mateo, CA.

24. Horowitz, M., Slamowitz, M., Rose, B. and Johnson, M., A 3.5ns, 1 Watt, ECL Register File, IEEE, *International Solid-State Circuits Conference*, San Francisco, CA. February, 1990.

25. Ligthart, M. M., Bechtolsheim, De Micheli, G. and El Gamal, A., Design of a Digital Audio Input Output Chip, IEEE, *Custom Integrated Circuits Conference*, San Diego, CA. May, 1989.

26. Martonosi, M. and Gupta, A., Tradeoffs in Message Passing and Shared Memory Implementations of a Standard Cell Router, *International Conference on Parallel Processing*, 88-96, August, 1989.

27. McFarling, S., Program Optimization for Instruction Caches, ACM/IEEE, *ASPLOS-III*, Boston, MA. April, 1989.

28. Nguyen, T. A. and Weise, D., Diagnosing Multiple Faults in Digital Systems., IEEE, *Artificial Intelligence Applications Conference*, Miami Beach, FL. March, 1989.

29. Nowick, S. M. and Dill, D. L., Practicality of State-Machine Verification of Speed-independent Circuits, *International Conference on Computer-Aided Design*, 1989.

30. Odani, M., Hwang, S. Y., Blank, T. and Rokicki, T. The Hermod Behavioral Synthesis System. *Journal of Systems and Software*. 1989.

31. Przybylski, S. *Performance-Directed Memory Hierarchy Design*. Ph.D., Stanford University, September, 1988. Also published as technical report No. CSL-TR-88-366.

32. Richardson, S. and Ganapathi, M., Interprocedural Analysis Vs. Procedure Integration, Info. Proc. Letters, April, 1989.

33. Richardson, S. and Ganapathi, M. Code Optimization Across Procedures. *IEEE Computer*. 22, (2): 42-50, February, 1989.

34. Richardson, S. and Ganapathi, M. Interprocedural Optimization: Experimental Results. *Software -- Practice and Experience*. 19, (2): 149-169, February, 1989.

35. Rose, J. Parallel Global Routing for Standard Cells. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*. 1989.

36. Rose, J., Klebsch, W. and Wolf, J. Temperature Measurement and Equilibrium Dynamics of Simulated Annealing Placements. *IEEE Trans. on Computer-Aided Design of Circuits and Sytems*. 1989.

37. Rose, J., Francis, R., Chow, P. and Lewis, D., The Effect of Logic Block Complexity on Area of Programmable Gate Arrays, IEEE, *Custom Integrated Circuits Conference*, 1989.

38. Rothberg, E. and Gupta, A., Experiences Implementing a Parallel ATMS on a Shared-Memory Multiprocessor, *International Joint Conference on Artificial Intelligence*, August, 1989.

39. Rothberg, E. and Gupta, A., *Fast Sparse Matrix Factorization on Modern Workstations*, Stanford University, Technical, October, 1989.

40. Ruf, E. and Weise, D., Nondeterminism and Unification in LogScheme: Integrating Logic and Functional Programming, *Conference on Functional Programming Languages and Computer Architecture*, London, England. September, 1989.

41. Salz, A. and Horowitz, M., IRSIM: An Incremental MOS Switch-Level Simulator, IEEE/ACM, *26th Design Automation Conference*, June, 1989.

42. Santoro, M. and Horowitz, M. SPIM: A Pipelined 64 X 64 Bit Interative Multiplier. *Journal of Solid State Circuits*. SC-24, (2): 487-493, April, 1989.

43. Santoro, M. *Design and Clocking of VLSI Multipliers*. Ph.D., Stanford University, October 1989, 1989. Also appears as CSL Technical Report no. 89-397.

44. Schnorf, P. and Ganapathi, M., *Compilation of Single Assignment Languages: Analysis and Propositions*, Stanford University, Technical, October, 1989. To be published.

45. Soule, L. and Gupta, A., Characterization of Parallelism and Deadlocks in Distributed Logic Simulation, IEEE/ACM, *26th Design Automation Conference*, Las Vegas, NV. June, 1989.

46. Soule, L. and Gupta, A., *Analysis of Parallelism and Deadlocks in Distributed-Time Logic Simulation*, Stanford University, Technical, CSL-TR-89-378, 1989. Submitted for publication.

47. Stark, D. and Horowitz, M. Techniques for Calculating Currents and Voltages in VLSI Power Supply Networks. *IEEE Transactions on Computer-Aided Design*. 1989.

48. Tamura, L., Yang, T.-S., Wingard, D., Horowitz, M. and Wooley, B., A 4-ns BiCMOS Translation Lookaside Buffer, IEEE, *International Solid-State Circuits Conference*, San Francisco, CA. February, 1990.

49. Tucker, A. and Gupta, A., Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors, ACM, *12th Symposium on Operating Systems Principles*, Lichtfield Park, AZ. December, 1989.

50. Weber, W. D. and Gupta, A., Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results, IEEE, *16th International Symposium on Computer Architecture*, Jerusalem, Israel. June, 1989.

51. Weber, W. D. and Gupta, A., Analysis of Cache Invalidation Patterns in Multiprocessors, IEEE/ACM, *ASPLOS-III*, Boston, MA. April, 1989.

52. Weise, D., Constraints and Multilevel Verification, *Workshop on Hardware Specification, Verification and Synthesis: Mathematical Aspects*, Ithaca, NY. July, 1989.

53. Williams, T. E. and Horowitz, M. Bipolar Circuit Elements providing Self-Completion-Indication. *Journal of Solid State Circuits*. August, 1989.

54. Wingard, D. E., Stark, D. C. and Horowitz, M. A. A 4ns 64Kb BiCMOS SRAM. 1989.

55. Wong, D., De Micheli, G. and Flynn, M., Designing High-Performance Digital Circuits Using Wave Pipelining, IEEE, *VLSI Conference*, Munich, W. Germany. August, 1989.

56. Rose, J. S., The Parallel Decomposition and Implementation of an Integrated Circuit Global R, ACM, *Sigplan Symposium on Parallel Programming*, 138-145, New Haven, CT. July, 1988.

57. Rose, J., Klebsch, W. and Wolf, J., Temperature Measurement of Simulated Annealing Placements, IEEE, *International Conference on Computer-Aided Design*, 514-517, 1988.

4. Staff

Faculty:
De Micheli, Giovanni
Dill, David
Gupta, Anoop
Hennessy, John (Principal Investigator)
Horowitz, Mark (Associate Investigator)
Lam, Monica
Meng, Teresa
Weise, Daniel

Research Staff:
Ganapathi, Mahadevan
Nakahira, D.
Przybylski, Steven
Rose, Jonathan

Graduate Students:
Anderson, J.
Basu, J.
Blatt, M.
Chanak, T.
Chandra, R.
Cho, Y.
Davis, H.
Gasbarro, J.
Gharachorloo, K.
Goldschmidt, S.
Ho, John Lee
Huyser, Karen
Johnson, W.
Jurvetson, S.
Larrabee, T.
Laudon, J.
Lenoski, D.
Lopez, R. Hernandez
Martonosi, M.
Mayden, D.
McFarling, S.
Pieper, K.
Richardson, S.
Rothberg, E.
Simoni, R.
Singh, J.
Smith, M.
Soule, L.
Stark, D.
Tjiang, S.
Todesco, A.
Tucker, A.
Weber, W.-D.
Williams, T.
Wing, M.
Wingard, D.
Wolf, M.